A SEARCH METHOD IN CONVEX PROGRAMMING

BY

MICHAEL ROGSON

UNIVERSITY OF CALIFORNIA AT BERKELEY

FEBRUARY 1965

AD 619 211

CLEARINGHOUSE FOR FEDERAL SCIENTIFIC AND TECHNICAL INFORMATION, CFSTI
INPUT SECTION 410.11

LIMITATIONS IN REPRODUCTION QUALITY OF TECHNICAL ABSTRACT BULLETIN DOCUMENTS, DEFENSE DOCUMENTATION CENTER (DDC)

☐ 1. AVAILABLE ONLY FOR REFERENCE USE AT DDC FIELD SERVICES. COPY IS NOT AVAILABLE FOR PUBLIC SALE.

[✓] 2. AVAILABLE COPY WILL NOT PERMIT FULLY LEGIBLE REPRODUCTION. REPRODUCTION WILL BE MADE IF REQUESTED BY USERS OF DDC.

[✓] A. COPY IS AVAILABLE FOR PUBLIC SALE.

☐ B. COPY IS NOT AVAILABLE FOR PUBLIC SALE.

☐ 3. LIMITED NUMBER OF COPIES CONTAINING COLOR OTHER THAN BLACK AND WHITE ARE AVAILABLE UNTIL STOCK IS EXHAUSTED. REPRODUCTIONS WILL BE MADE IN BLACK AND WHITE ONLY.

TSL-121-2 65

DATE PROCESSED: 8-23-65

PROCESSOR: [illegible]

A search method in convex programming

by

Michael Rogson

Technical Report
Prepared under Contract Nonr 222(85)
Office of Naval Research

Department of Mathematics
University of California

February 1965

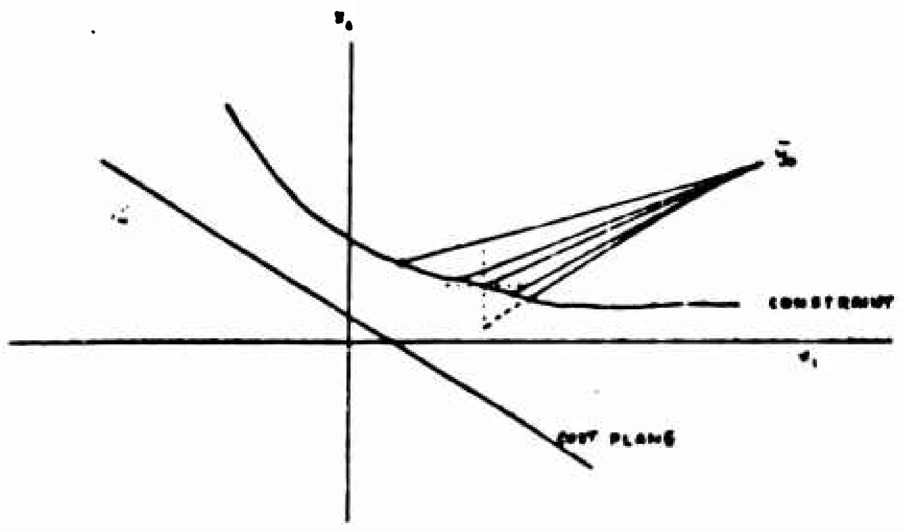
TABLE OF CONTENTS

Introduction

Before describing our method, we first recall the basic convex programming problem: Given a convex set in n-dimensional Euclidean space (in the following called <u>constraint set</u>). Given a hyperplane (referred to as the <u>cost plane</u>) not intersecting the constraint set. The problem: find a point in the set such that the distance from the hyperplane is minimal compared with all other points in the set. Any point that is in the set but not necessarily the closest to the cost plane is called a <u>feasible solution</u>.

Our method is an <u>evolution-process</u>. We assume that we have an initial feasible solution $\bar{y}_o$. The procedure is to subject $\bar{y}_o$ to processes abstracted from biological evolution: "mutation", "mating" and "selection of the fittest".

By mutation we mean the following: Given a point $\bar{x}$ on the boundary of the constraint set, we consider the vector $\bar{d} = \bar{y}_o - \bar{x}$. The coordinates of $\bar{d}$ may be thought of as corresponding to "genes". We "mutate" $\bar{d}$ by adding a perturbation vector all of whose coordinates are zero except one (we are mutating one gene at a time).



By "mating" we understand the following: given points $\bar{x}$, $\bar{z}$ we have $\bar{d}_x = \bar{y}_o - \bar{x}$ and $\bar{d}_z = \bar{y}_o - \bar{z}$ as the corresponding directions. Let $\bar{d} = (\bar{d}_x + \bar{d}_z)/2$, then we consider the point $\bar{u}$ on the constraint surface and the line with direction $\bar{d}$ through $\bar{y}_o$ to be the "offspring" of the "mated parents", $\bar{x}$, $\bar{z}$.

"Selection of the fittest" means that we choose the vector whose cost is smallest from the family we generated. That is to say, we choose from all the specimens generated the optimum one. If this optimum is an improvement over the last one in the cycle, we return for another one, providing we have not cycled back more than a predesignated number of times.

We will first describe and analyze the method and its implementation for linear constraint functions - linear programming. Then we proceed to discuss the method for the more general problem with non-linear constraints. The process is started by taking as first $\bar{x}$ the intersection point of the line through $\bar{y}_o$ that is perpendicular to the cost plane (the gradient direction).

I would like to thank the many persons who were so generous with their help during all stages of this research. In particular, my thanks to Professor H. J. Bremermann for his help and encouragement as committee chairman; to Professors G. B. Dantzig and D. H. Lehmer for their suggestions as committee members; to Steven Salaff, who had many constructive criticisms during tive criticisms during the early stages of this work. Finally, I also want to express my gratitude to the Office of Naval Research, without whose support this research would not have been possible, and to C.E.I.R., Inc., who made available their machine on many occasions.

Description of the Method

The problem that we are trying to solve is the following: We are given a system of m linear inequalities in n unknowns with real coefficients:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &\geq b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &\geq b_2 \\ \vdots \qquad \vdots \qquad & \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &\geq b_m \end{aligned}$$

These inequalities shall also be called <u>constraints</u>. To simplify our notation we shall represent the constraints in matrix form:

$$Ax \geq b$$

where:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & & a_{mn} \end{pmatrix}$$ is the matrix of coefficients

and

$$b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

We are also given a linear <u>cost function</u>

$$cx = c_1x_1 + c_2x_2 + \cdots + c_nx_n = z$$

We note that this linear cost function defines a hyperplane in n-dimensional Euclidean space. Similarly the equations

$$a_{11}x_1 + \dots + a_{1n}x_n = b_1$$

also define hyperplanes which bound the convex polyhedron. We say that a vector $x = (x_1, \dots, x_n)$ is feasible if $Ax \leq b$. That is to say x satisfies the constraints. x is called optimal if x is feasible and for any other feasible vector y $cx \leq cy$. Our problem is to find such an optimal vector x, providing it exists.

Before we describe our method in detail we will briefly outline it in an intuitive manner. Our method assumes a given initial feasible vector from which we can start. Denote this vector by y_0, for a point in n-space. From this point we travel in the direction perpendicular to the cost hyperplane till we reach the boundary of the constraint polyhedron. After this point, whose coordinate vector we denote by x, has been found, we proceed to "shoot a buckshot volley" in the direction $x-y_0$. By "buckshot volley" we understand the following: From a point inside the polyhedron we proceed in several random trial directions that are contained in a circular cone of some given solid angle around the given fixed direction. The distribution of the random directions is analogous to the trajectories traced by individual grains of a blast of buck shot. We then determine the intersection points of the trajectories with the polyhedron. The coordinate vectors of these intersection points are feasible

The coordinate vectors of these intersection points are feasible vectors since they satisfy $Ax \leq b$ (with the equal sign occurring in at least one component). Among these vectors there is, we hope, one vector x' having a cost that is less than the cost of x and also such that the norm of the difference between x and x', i.e. $\sum_{i=1}^{n} (x'_i - x_i)^2$ is greater than a certain ε which we will accept as the smallest change in vectors.

If the norm of the difference is greater than ε and we have not exhausted a given maximum number of loops we then exchange x' and x and shoot another "volley" around the new direction determined by the new x and y_0. This method is continued until we either are not improving our approximation fast enough, that is $\Sigma(x'_i - x_i)^2 \leq \varepsilon$, or we exceeded the maximum number we allowed for iterations.

In what follows we shall try to describe the method we are using to effect the above mentioned "buckshot" technique.

To simplify the description we want to introduce some further notation: by e_i we will understand the vector defined by the point in our n-dimensional Euclidean space that has a 1 in the i<u>th</u> coordinate and 0's elsewhere. In other words: $e_i = (\delta_{i1}, \delta_{i2}, \cdots, \delta_{ij}, \cdots, \delta_{in})$ where $\delta_{ij} = \begin{cases} 1 & \text{if } i=j \\ 0 & \text{if } i \neq j \end{cases}$ is Kronecker's delta.

For the "buckshot" effect we need to have some dimensions for the cone which will determine how "wide" or "open" the buckshot will be. For this we read in two factors h, ℓ, a high and a low factor respectively. The factor h will be multiplied by a random number, uniformly distributed between 0 and 1, during the course of each buckshot. The factor ℓ will be set to $\frac{1}{2}\sum_{i=1}^{n}(x_i'-x_i)^2$ if $\ell < \sum_{i=1}^{n}(x_i'-x_i)^2$, to reduce the small cone and increase our rate of convergence as the approximation comes closer to a limit point.

The process starts as follows: we generate a set of feasible vectors by modifying the direction given by y_0 and x, our initial feasible vector and our current best approximation respectively, by adding to it:

$$d = y_0 - x + f\, e_i \qquad \text{for } i = 1,\cdots, n$$

when f is h or ℓ, the high and low factors respectively. This gives us for each of the n-dimensions 4 new vectors and since we include x in the family we have $4n+1$ feasible vectors. These vectors, which we shall call the "parents", are when "pairwise mated". If we denote by x^j the members of the parents set, $j=1,\cdots,4n+1$, then by pairwise mating we mean the following: We determine the coordinate vector z of the intersection of the lines through y_0 in the direction

$$d = y_0 - 0.5(x^j + x^k) \quad \text{where } 1 \le j \le 4n+1$$
$$1 \le k \le 4n+1 \text{ and } j \ne k$$

with the boundary of the constraint polyhedron. By pairwise mating we thus obtain

$\binom{4n+1}{2} = 2n(4n+1)$ vectors. We note that for large values of n this method implies a prohibitively large number of operations. To somewhat counteract this problem we have introduced a modification which will enable us to skip over a number of parents in the "mating" process. We have thus been able to analyze some higher dimensional cases.

By ranking this family of vectors according to the cost we can proceed to find the one with lowest cost function value and proceed with the iteration method described above.

We shall now give a description of how the method works in a step by step manner:

1. Given the initial feasible vector y_0, the constraints $Ax \geq b$ and the cost function $cx = z$, our first step is to find the vector x in the direction normal to the plane defined by the cost function. This vector x now becomes the first vector in our scheme and the iteration cycle begins

2. We take the direction determined by y_0, our initial feasible vector, and our current best approximation vector x:

$$d = y_0 - x = (y_{0_1} - x_1, \cdots, y_{0_n} - x_n)$$

and effect our buckshot effect around this direction.

3. From the family generated in step 2 we pick the vector x' which has the least cost and we evaluate

$$t = |x'-x| = \sqrt{\sum_{j=1}^{n} (x_j'-x_j)^2}$$

> If $t > \varepsilon$ then we proceed to exchange x and x' and check whether this was the last permissible iteration. We next proceed to check if our running low factor for the buckshot effect, ℓ, is less than t, if it is then we set $\ell = 1/2\ t$, otherwise we leave it the same. Then, if we have not exceeded the maximum number of iterations we return to step 2. If not we finish this run by saving the information gathered so far. On the other hand if $t < \varepsilon$ then we assume that our improvement is too slow and we also proceed to save the information and exit.

We have so far repeatedly mentioned that we travel along a given direction through the initial feasible vector until we hit a face of the polyhedron, we shall now proceed to describe the method by which we do this and also give the mathematical justification for it:

Let us denote by d the given direction in which we want to travel starting from y_0. And let Ry_0 be the non-negative vector $b - Ay_0$.

$$\lambda = -\min_{A_i d<0}\left(\frac{-Ry_{0_i}}{A_i d}\right) \quad \text{if for } i = 1,\cdots,m \quad A_i d < 0$$

or

$$\lambda = \min_{A_i d>0}\left(\frac{Ry_{0_i}}{A_i d}\right) \quad \text{if for some } i \quad A_i d > 0$$

where $\min\limits_{A_i d>0}\left(\frac{Ry_{0_i}}{A_i d}\right)$ means we take the smallest positive

$\frac{Ry_{0_i}}{A_i d}$ for $i = 1,\cdots,n$.

We will show that the vector $x = y_0 + \lambda d$ is a feasible vector and for some i we have the condition $A_i x = b_i$, which means that x is on the hyperplane (we will say "face" for short) defined by that equation.

Proof. We first consider the case when for some

i $A_i d > 0$. Then

$$\lambda = \min_{A_i d > 0} \left(\frac{Ry_{0_i}}{A_i d} \right) = \frac{Ry_{0_{i_0}}}{A_{i_0} d}$$

then $Ax = A(y_0 + \lambda d) = Ay_0 + \lambda Ad$

" $Ax = Ay_0 + \left(\frac{Ry_{0_{i_0}}}{A_{i_0} d} \right) Ad$

That is for $j = 1, \cdots, m$

$$A_j x = A_j y_0 + \left(\frac{Ry_{0_{i_0}}}{A_{i_0} d} \right) A_j d$$

now for $j = i_0$, $A_{i_0} x = A_{i_0} y_0 + \left(\frac{Ry_{0_{i_0}}}{A_{i_0} d} \right) A_{i_0} d = A_{i_0} + Ry_{0_{i_0}}$

since $Ry_{0_{i_0}} = b_{i_0} - A_{i_0} y$ we have

$$A_{i_0} x = A_{i_0} y_0 + b_{i_0} - A_{i_0} y_0 = b_{i_0}$$

thus showing for some i_0 $A_{i_0} x = b_{i_0}$

Now for $j \neq i_0$ we have

1. if $A_j d \leq 0$

then $A_j x = A_j y_0 + \left(\frac{Ry_{0_{i_0}}}{A_{i_0} d} \right) A_j d \leq A_j y_0 \leq b_j$

hence $A_j x \leq b_j$

2. if $A_j d > 0$ then $\frac{Ry_{0_j}}{A_j d} > \frac{Ry_{0_{i_0}}}{A_{i_0} d} > 0$

thus $A_j x = A_j y_0 + \frac{Ry_{0_{i_0}}}{A_{i_0} d} A_j d \leq A_j y_0 + \frac{Ry_{0_j}}{A_j d} A_j d$.

$$A_j x \leq A_j y_0 + Ry_{0_j} = A_j y_0 + b_j - A_j y_0 = b_j$$

Hence if $A_j d > 0$ for some j we have shown our claim. For the case $A_j d \leq 0$ for $j = 1,\cdots,m$ we can consider exactly the same situation as above with the distinction that we multiply d by -1 and then subtract rather than add λd. This can be done since d is nothing but an ordered set of directions numbers and by multiplying by -1 we have not changed the direction except for the sense in which we were traveling on the line given by the direction d and the vector y_0.

For the initialization of the technique we give as a direction vector the normal to the cost plane, i.e., the plane defined by the linear cost function c. We do this in order to get an "optimal" start to the method since further directions are going to be defined by the succeeding approximations together with the given initial feasible vector y_0.

Before we go into discussing the experimental data obtained from the method discussed above we will describe a few changes which were introduced during the course of the experimentation:

1. On page 6 we defined

$$\lambda = -\min_{A_i d<0}\left(\frac{-Ry_{0_i}}{A_i d}\right) \text{ if for } i = 1,\cdots,m \quad A_i d \leq 0$$

or

$$\lambda = \min_{A_i d > 0}\left(\frac{Ry_{0_i}}{A_i d}\right) \text{ if for some } i \quad A_i d > 0$$

We found that if we let

$$\lambda_1 = -\min_{A_i d<0}\left(\frac{-Ry_{0_i}}{A_i d}\right) \quad \text{and}$$

$$\lambda_2 = \min_{A_i d>0}\left(\frac{Ry_{0_i}}{A_i d}\right) \quad \text{for } i = 1,\cdots,m$$

and define

$$\lambda = \min_{\lambda k} (\lambda_1 \, cd, \lambda_2 \, cd) \quad \text{when } k = 1,2$$

then we know that we will not only travel along the direction given by the d vector until we hit a face of the polyhedron but we will minimize the cost in the subspace defined by d. This modification did not affect the rate of convergence at all among the analyzed problems.

2. We recall that the "buckshot" effect was accomplished by generating a set of "parents" by (asexual) mutations on our last best rector. These parents are then mated in a pairwise manner. The change we introduced consists of allowing to skip

ahead the generation of the parents and the pairwise mating, if during the asexual mutations we found a mutant which has an improvement upon the running best vector that is greater than a certain ε we read in as input data. That is, if we find a mutant whose cost is smaller than the cost of the last vector minus ε, then we consider the mutant the best vector, as if we were of the end of a complete iteration and continue at the beginning of the next one by considering the "asexual" improvement the next best vector.

This change, seemingly minor in character allowed for a tremendous upsurge in the time rate of improvement -- that is, timewise the convergence was greatly accelerated. Because of this we were able to handle problems with almost complete success which previously were not feasible. Furthermore in those cases where the unmodified version worked well there was also an increase in the accuracy of convergence.

3. A third change that was suggested, for e buckshot effect was not to "mate" parents which lie on the same hyperplane. That is, if we let x and z denote two arbitrary parents we would not mate them if for some i

$$b_i - A_i x = 0 = b_i - A_i z$$

This change does not seem to affect the experiements in any measurable way.

4. The last suggested change was to move the original vector y_0 from its static location to a more centralized location. Several different methods were attempted none of which showed any improvement on preliminary tests.

A CLASS OF LINEAR PROGRAMMING PROBLEMS

Next we shall describe the class of problems that was used for the experimentation in the linear programming part: Let us denote by I the n-dimensional identity matrix

$$I = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}$$

by J the $n \times n$ matrix consisting of '1's

$$J = \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{pmatrix}$$

and by b the n-dimensional column vector consisting of 1's

$$b = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

Further, let us define $A = \alpha I + J$ where α is some positive real number. The class of constraint inequalities was $Ax \geq b$, with a linear objective function; the cost function $cx = \Sigma x_i$, i.e., $c = (1, \cdots, 1)$, the n-dimensional row vector consisting of ones.

The main advantage in using these constraints lies in their handling ease, as will be obvious in the sequel. The original suggestion to use these matrices came from Hooke and Jeeves, [15], who developed an interesting search technique for the solution of linear equations.

By the <u>condition number</u> of a matrix we understand the ratio of the largest to the smallest eigen value of the matrix. To find the eigen values of $\alpha I + J$ we note that $|\alpha I_n + J_n - \lambda I_n| = |(\alpha - \lambda)I_n + J_n|$ where the subscript n is used to denote the dimensionality of the determinants.

$$|(\alpha-\lambda)I_n + J_n| = \begin{vmatrix} (\alpha-\lambda)+1 & \cdots & & 1 \\ 1 & (\alpha-\lambda)+1 & 1 \cdots & 1 \\ \vdots & & & \vdots \\ 1 & \cdots & & (\alpha-\lambda)+1 \end{vmatrix}$$

$$= (\alpha-\lambda+1) \begin{vmatrix} \frac{1}{\alpha-\lambda+1} & \frac{(\alpha-\lambda+1)^2-1}{\alpha-\lambda+1} & \cdots & \frac{(\alpha-\lambda+1)\ -1}{\alpha-\lambda+1} \\ \vdots & & & \vdots \\ \frac{1}{\alpha-\lambda+1} & & & \frac{(\alpha-\lambda+1)^2-1}{\alpha-\lambda+1} \end{vmatrix}$$

$$= \frac{(\alpha-\lambda+1)}{(\alpha-\lambda+1)^{n-1}} \cdot (\alpha-\lambda)^{n-1} \begin{vmatrix} (\alpha+1)-\lambda+1 & 1 \cdots & 1 \\ 1 & (\alpha+1)-\lambda+1 & \vdots \\ \vdots & & \\ 1 & \cdots & (\alpha+1)-\lambda+1 \end{vmatrix}$$

$$= \frac{(\alpha-\lambda)^{n-1}}{(\alpha-\lambda+1)^{n-2}} \left|((\alpha+1)-\lambda)I_{n-1} + J_{n-1}\right|$$

$$= \frac{(\alpha-\lambda)^{n-1}}{(\alpha+1-\lambda)^{n-2}} \cdot \frac{(\alpha+1-\lambda)^{n-2}}{(\alpha+2-\lambda)^{n-3}} \cdots \frac{(\alpha+n-3-\lambda)^2}{(\alpha+n-2-\lambda)} \cdot \left|((\alpha+n-2)-\lambda)I_2 + J_2\right|$$

$$= (\alpha-\lambda)^{n-1}(\alpha-\lambda+n)$$

Hence the eigen values are α, $n-1$ times, and $\alpha+n$ once. The condition number is thus found to be $c = \frac{\alpha+n}{\alpha}$ which can be readily changed by manipulating α. The solution of the problem is $x = A^{-1}b = \left(\frac{1}{\alpha}I - \frac{1}{\alpha(n+\alpha)} J\right) b$.

We shall first report the experiments that were carried out with the original method described on page 5 and thereafter we will discuss the effect of the modifications described on page 9 .

Rather than discussing all the problems that we experimented with we shall discuss two typical examples and only on occasion comment on others as the need arises.

For the five dimensional case, i.e., $n = m = 5$, and with a condition number of 5 we observed that the method converges. The convergence, however, is strongly dependent on the value of the dimensions of the cone of the buckshot effect. For example, if we start with an $f_h = 2.0$ and an $f_l = 0.08$, where f_h and f_l are the high and low arguments respectively, the method iterates 11 times and then finds no improvement among the buckshot, i.e., $x = x'$. The result remained, after 11 iterations with a maximum error of 1/10. The same problem run with $f_h = 1.0$ and $f_l = 0.1$ converged beyond the point where the small component factor started to be modified.

For the analogous problem with $n = m = 5$ but with a condition number of 1.19 we find the same behavior except that the error for this case came out to be 1/1000 after 20 iterations.

It is interesting to note that this problem also crops up when we deal with higher dimensional cases. We want to note that the same problem of cones that are too large for the buckshot effect was dramatized in the case for $n = 30 = m$ with a condition number of 2, when after 57 iterations we terminated the execution since no improvement was found in the family generated by f_h and f_l. They were $f_h = 1$ and $f_l = 0.1$. This leads one to speculate that the perturbation factors f_h and f_l may be inversely related to the dimension of the system.

Returning now to our discussion of the 5-dimensional case we would like to discuss the effect of changing the original feasible vector in such a way that it is no longer so "neatly" centralized. We noted that although the method empirically converges its rate of convergence is slower, namely when:

$$y_o = (2.5, 9, 5.5, 3\ 14) \text{ as opposed to}$$

$$y_o = (3.5, 2, 4, 3, 5)$$

in 15 iterations the largest error was 2/100 whereas the largest error at the end of 20 iterations of centralized y_o was 1/1000.

As the method was originally programmed for a computer it was not practical for larger systems at all, hence we introduced a modification which would make the buckshot family smaller. We recall that by making 2 large and 2 small perturbations for each coordinate direction we have that the number of vectors in the "parent" set is 4n. We then proceeded to pairwise mate this set, including also the best running approximation we thus get $\binom{4n+1}{2} = 2n(4n+1) = 8n^2 + 2n$. For $n = 100$ this makes the family have 80288 vectors. To generate such a large family is infeasible because of time considerations. We should mention that when we tried the full-blown scheme explained above for $n = 30$ it took 35 minutes on the IBM 7094 to iterate just 9 times. The way we have gotten around this difficulty is by skipping over a goodly number of the parents in our mating. When this was done, the time was considerably reduced but the rate of convergence per major cycle for those experiments tried, was not reduced in any marked way. For example, for $n = 5$ we tried skipping over every 2 "parents". The number in the total family was reduced by a factor of 4 and the rate of convergence was not modified at all. In higher dimensions, for the cases tried, this accelerated the convergence rate.

For one particular run on the 5-dimensional case we also set up a problem where we had more constraints than variables. In this problem again the method empirically converged. The case we tried is:

$$\begin{pmatrix} 2.25 & 1 & 1 & 1 & 1 \\ 1 & 2.25 & 1 & 1 & 1 \\ 1 & 1 & 2.25 & 1 & 1 \\ 1 & 1 & 1 & 2.25 & 1 \\ 1 & 1 & 1 & 1 & 2.25 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

and $cx = x_1 + x_2 + x_3 + x_4 + x_5$.

The estimated solution vector converged to

$x = (1, -23, 1, -23, -24)$ after 15 iterations.

As we mentioned above we also tried some higher dimensional cases. A 9-dimensional problem converged to the answer with a maximum error of 2.5/1000 in 30 iterations. We ran a 30-dimensional case that had to be stopped as a failure after 46 iterations. This took about seven minutes. If we extrapolate from our analysis of smaller dimensional cases the convergence seemed to be on the right track and, apparently the size of the cone was too large in our buckshot effect.

Let us now investigate the consequences of the changes introduced on page 9 by comparing the results of identical runs with the modified and unmodified versions of our method. We want to note that the modification was introduced in such a manner that it was possible to use the same program for both versions by simply changing an input constant τ. To accept or reject "asexual" improvements depends on this τ which when large enough would reject all "asexual" improvements. We want to note that the method was somewhat streamlined by more efficient programming. This helped in reducing running time, as can be observed by the time estimates.

To begin with, we will start with the 5-dimensional case, which is the one most extensively investigated:
For a condition number of 2.25 the old version ran for 3.6 seconds and 6 loops to stop with a maximum error or 0.1128. This error seems largely due to the large cone factor for the buckshot effect. The modified version, however, ran for 10.8 seconds and 17 loops to stop with a maximum error of 0.00026. The maximum error after 6 loops in the second run was 0.0587;

Next let us consider the cases for a condition number of 5.07. With a large cone factor the unmodified version in 4.8 seconds looped 6 times and terminated with a maximum error of 0.009971; its associate looped 10 times in 7.2 seconds and terminated with an error of 0.000045.

By changing the constant which limits the asexual improvements also note some change in the rate of convergence to the solution. For example, by letting $\tau = 0.1$, the high cone factor $f_h = 2.$, and the small cone factor $f_\ell = 0.1$, we obtained in 1.8 seconds 4 loops with an error of 0.067. By changing $\tau = 0.01$ and the high cone factor to 1, we iterate 10 times for 4.8 seconds with a maximum error of 0.000035. By further reducing τ to zero in 3 seconds we iterate 6 times and terminate with an error 0.000021 -- in this last run the high cone factor was 2.0. Now when we changed the cone factor to 1 we found that we terminate in the second loop with a maximum error of 0.017. This pattern continues when we change the condition numbers of the matrices we experiment with. However for one case with a condition number of 10 we find that with a high cone factor of 1 the unmodified version in 6.6 seconds loops 11 times and terminates with a maximum error of 0.000256. With a τ of 0.01 we iterate for 10.2 seconds and 16 loops to end with an error of 0.006666, and for $\tau = 0$ in 2.4 seconds and 3 loops we find a maximum error of 0.001446! With a different set of random numbers, however, the above described problem ran 4.8 seconds and 7 loops with a maximum error of 0.000014. The same pattern continues for condition numbers of 15 and 21. We refer to the summary table 2 at the end of this discussion for a global picture of all the runs we made for the 5 dimensional problem.

Let us now investigate the 9 dimensional problems we experimented with. For a condition number of 2.25 we found that:

If we let the cone factor be 0.1 the old version iterates in one minute and 55.8 seconds 77 times and terminated with an error of 0.044760. When the new version was used, with an asexual improvement τ of 0.01 we find that in 32.4 seconds and 13 loops the maximum terminal error is 0.017109. When we used a cone factor of 2 and $\tau = .01$ we find that in 10 iterations that took 25.8 seconds the error is 0.036451. By reducing to $\tau = 0$ we have in 7 iterations and 22.2 seconds a maximum error of 0.000044, with a large cone factor of 0.1. For higher condition numbers we experimented with a ten dimensional case, viz., we considered a condition number of 5. When the high cone factor was 2 and we used the older version we find that in 26 iterations and 2 minutes we achieved a maximum error of 0.045106. When we permitted asexual improvements of 0.01 we find that the process is terminated in 41.4 seconds and 8 loops with an error of 0.010270. When we had a cone factor of 1.0 we iterate 22 times in 92.4 seconds and terminate with an error of 0.002213. When we try to reduce the family size by 16 we find that the same problem as above finds no improvements after 4.2 seconds and 2 loops and terminates with a maximum error of 0.134343. It is worth mentioning that the preceding experiment

Table 1, page 25, summarizes our original results with the unmodified version. Table 2, pages 26 through 28, summarizes the comparison of the two methods. The flow chart which follows the tables corresponds to the modified program. We did not include the flow chart of the unmodified version because we do not believe that the addition would add any valuable information.

TABLE 1

Dimension	Condition Number	Number of Iterations	Largest Error	Large Cone Factor	Small Cone Factor	Modification of Small Cone Factor	Time to Execute	k Step	ℓ Step
n=3, m=3	5.0	3	0.001	2.00	0.08	0.08	not avail	1	1
n=5, m=5	5.0	5	0.25	2.00	0.08	0.08	not avail	1	1
n=5, m=5	1.9	11	0.21	1.00	0.10	0.10	7.5 sec.	2	2
n=5, m=5	5.0	10	0.059	1.00	0.10	0.10	7.5 sec.	2	2
n=5, m=5	5.0	15	0.020	2.0	.08	0.08	not avail	1	1
n=5, m=5	2.0	20	0.002	1.0	0.10	0.004	15 sec.	1	1
n=5, m=5	5.0	19	0.0001	1.0	0.10	0.000146	15 sec.	1	1
n=5, m=5	2.0	22	0.005	2.0	0.08	0.0024	not avail	1	1
n=5, m=5	5.0	17	0.00045	2.0	0.08	0.0014	not avail	1	1
n=5, m=7	---	14	---	2.0	0.08	0.016	not avail	1	1
n=9, m=9	2.25	10	0.208	2.0	0.08	0.08	30 sec.	1	1
n=9, m=9	2.25	9	0.07	2.0	0.08	0.08	not avail	1	1
n=9, m=9	2.25	30	0.0025	2.0	0.08	0.00227	1.5 min.	1	1
n=10, m=10	2.85	21	0.077	0.8	0.05	0.05	not avail	1	1
n=20, m=20	4.33	218	20.24	0.8	0.05	0.0017	15 min.	5	5
n=20, m=20	4.33	316	15.45	0.8	0.05	0.00046	20 min.	5	5
n=30, m=30	2.00	30	14.50	1.00	0.10	0.10	2.4 min.	15	15
n=30, m=30	2.00	30	14.50	1.00	0.10	0.10	2.4 min.	10	10

TABLE 2 (Page 1)

Dimension	Condition Number	Number of Iterations	Largest Error	Large Cone Factor	Small Cone Factor	Modified Small Cone Factor	Time in Seconds	k Step	ℓ Step	τ
n=5, m=5	2.25	6	0.101978	2.0	0.1	0.1	3.6	1	1	300.
n=5, m=5	2.25	17	0.002678	2.0	0.1	0.003309	10.8	1	1	0.1
n=5, m=5	2.25	8	0.110974	2.0	0.1	0.1	1.2	3	3	300.
n=5, m=5	2.25	9	0.087806	2.0	0.1	0.1	1.2	3	3	0.1
n=5, m=5	2.25	19	0.009308	2.0	0.1	0.004663	2.4	3	3	300.
n=5, m=5	2.25	5	0.193042	2.0	0.1	0.1	1.2	3	3	0.1
n=5, m=5	5.0	5	1.259371	1.0	0.1	0.1	1.2	2	2	300.
n=5, m=5	5.0	17	1.074758	1.0	0.1	0.000843	2.4	2	2	300.
n=5, m=5	5.07	6	0.009971	2.0	0.1	0.03141	4.2	1	1	300.
n=5, m=5	5.07	10	0.000045	2.0	0.1	0.000107	7.2	1	1	0.01
n=5, m=5	5.07	4	0.057463	2.0	0.1	0.1	1.8	1	1	0.1
n=5, m=5	5.07	10	0.000035	2.0	0.1	0.000057	4.2	1	1	0.01
n=5, m=5	5.07	6	0.000021	2.0	0.1	0.00011	3.0	1	1	0
n=5, m=5	5.07	2	0.005353	2.0	0.1	0.01335	1.8	1	1	0
n=5, m=5	5.07	5	0.009971	2.0	0.1	0.03141	2.4	1	1	0.1
n=5, m=5	10.	11	0.000256	1.0	0.1	0.0000527	6.6	1	1	300.
n=5, m=5	10.	16	0.003666	1.0	0.1	0.000129	10.2	1	1	0.01
n=5, m=5	10.	3	0.001443	1.0	0.1	0.009276	2.4	1	1	0.0
n=5, m=5	10.	7	0.014429	1.0	0.1	0.000133	2.4	1	1	0.01

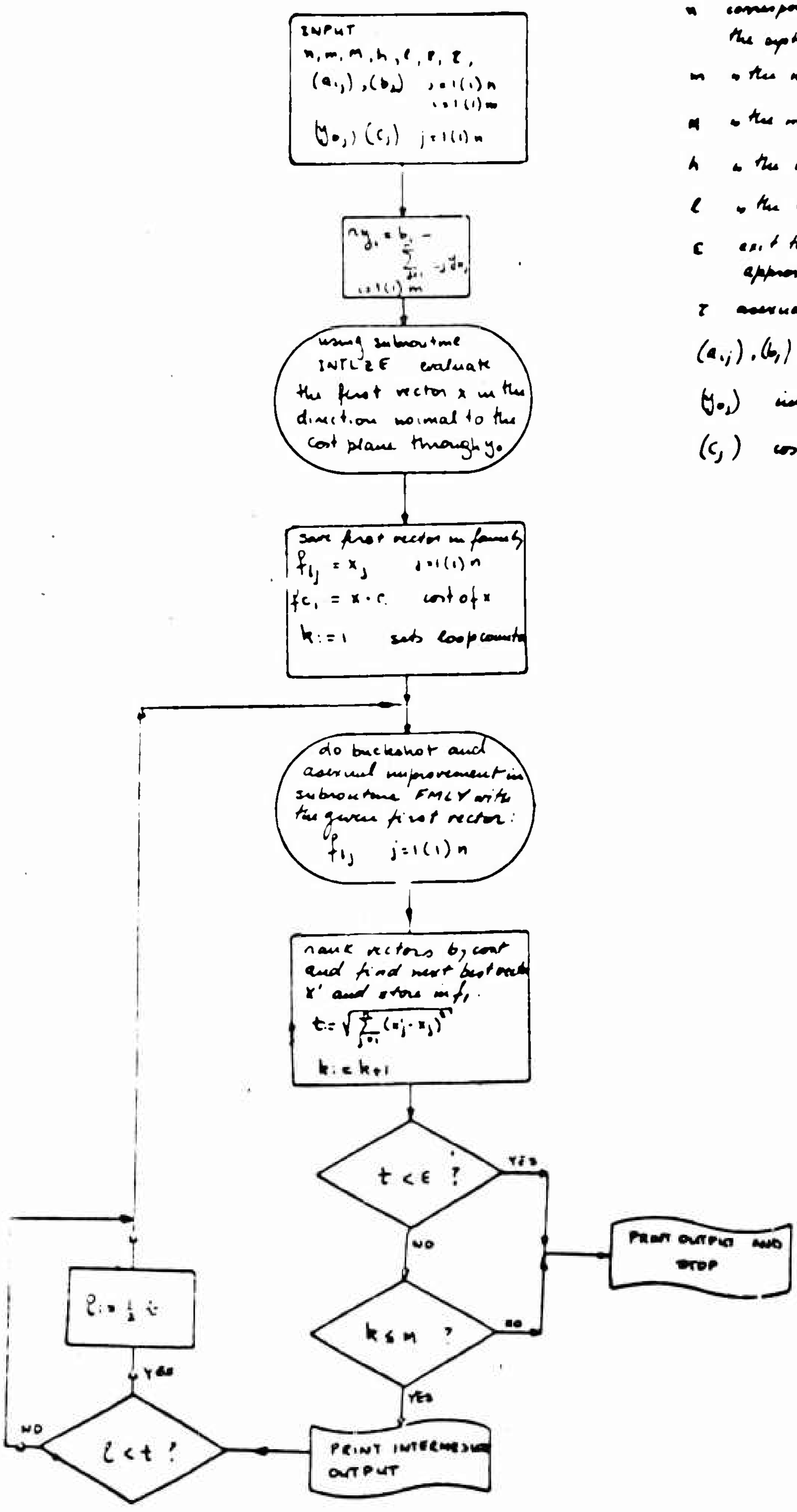
TABLE 2 (Page 2)

Dimension	Condition Number	Number of Iterations	Largest Error	Large Cone Factor	Small Cone Factor	Modified Small Cone Factor	Time in Seconds	k Step	ℓ Step	τ
n=5, m=5	10.	3	0.000855	1.0	0.1	0.0007625	2.4	1	1	0
n=5, m=5	10.	9	0.012022	1.0	0.1	0.00003904	4.2	1	1	0.01
n=5, m=5	10.	8	0.002418	2.0	0.1	0.0001385	4.8	1	1	0.01
n=5, m=5	10.	7	0.000014	1.0	0.1	0.0000574	4.8	1	1	0
n=5, m=5	15.7	13	0.018478	2.0	0.08	0.00001075	3.6	1	1	0.01
n=5, m=5	15.7	10	0.013072	1.0	0.05	0.92×10^{-5}	3.6	1	1	0.01
n=5, m=5	15.7	4	0.000001	2.0	0.08	0.00001435	6.0	1	1	0
n=5, m=5	15.7	3	0.000319	1.0	0.05	0.003	2.4	1	1	0
n=5, m=5	21.0	10	0.011420	1.0	0.1	0.00005833	6.0	1	1	300.
n=5, m=5	21.0	10	0.013692	1.0	0.1	0.0001894	6.0	1	1	0.1
n=5, m=5	21.0	11	0.002199	1.0	0.1	0.0001455	2.4	1	1	0.1
n=5, m=5	21.0	7	0.018936	1.0	0.1	0.0002217	2.4	1	1	0.1
n=5, m=5	21.0	33	0.002714	1.0	0.1	0.00005064	9.0	1	1	0.1
n=9, m=9	2.25	77	0.044760	0.1	0.05	0.67492	115.8	1	1	2.0
n=9, m=9	2.25	10	0.036451	2.0	0.08	0.00001285	25.8	1	1	0.01
n=9, m=9	2.25	13	0.026216	1.0	0.05	0.00000624	32.4	1	1	0.01
n=9, m=9	2.25	11	0.377216	2.0	0.08	0.0000076	66.0	1	1	0.01
n=9, m=9	2.25	16	0.030207	1.0	0.05	0.0000116	105.6	1	1	0.01
n=9, m=9	2.25	7	0.000044	0.1	0.05	0.00001503	22.2	1	1	0

TABLE 2 (Page 3)

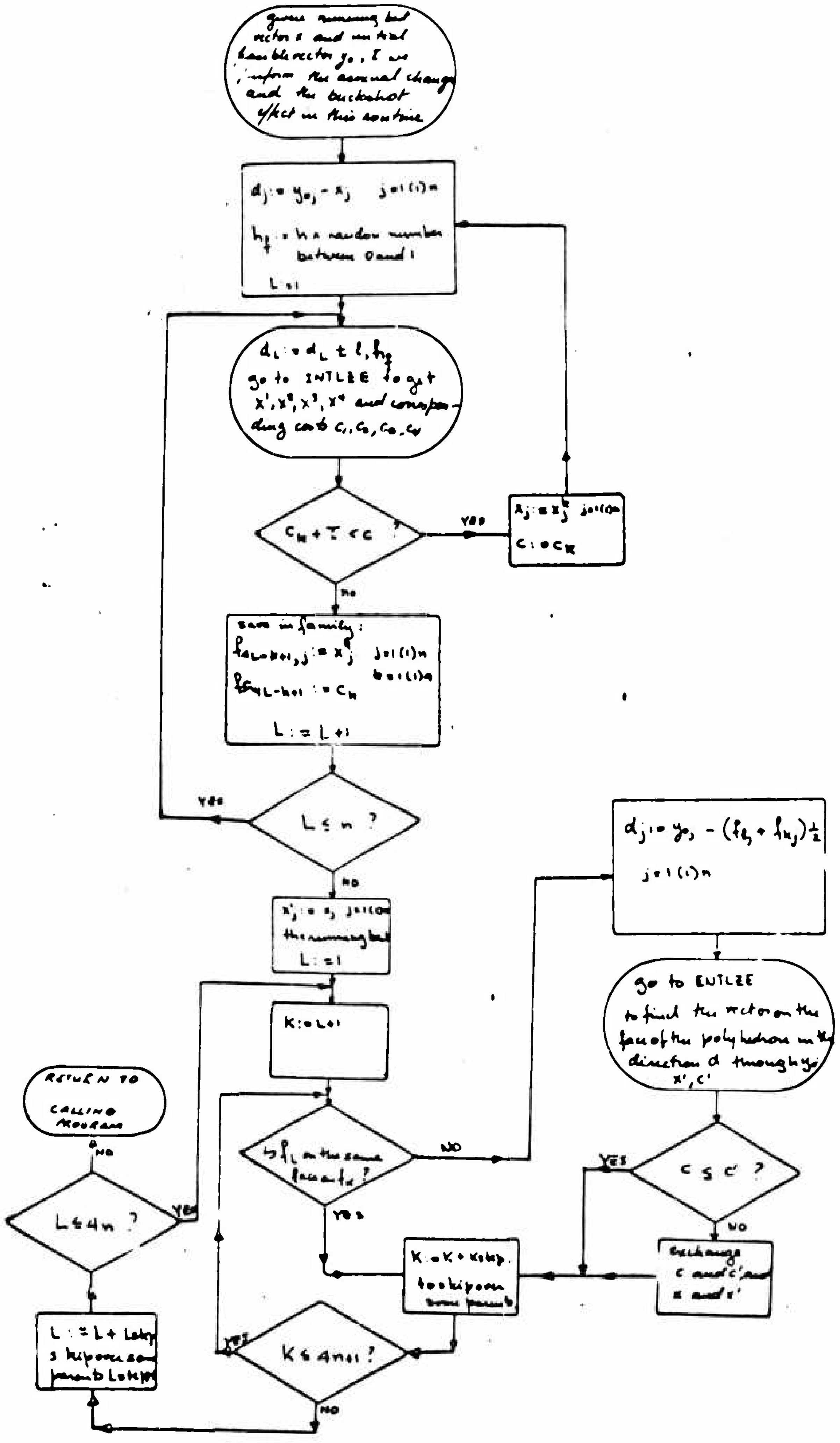
Dimension	Condition Number	Number of Iterations	Largest Error	Large Cone Factor	Small Cone Factor	Modified Small Cone Factor	Time in Seconds	k Step	ℓ Step	τ
n=9, m=9	2.25	9	0.000027	2.0	0.08	0.000088	25.2	1	1	0
n=10, m=10	5.0	23	0.045106	2.0	0.1	0.000509	119.4	1	1	300.
n=10, m=10	5.0	8	0.010270	2.0	0.1	0.00927	41.4	1	1	0.01
n=10, m=10	5.0	22	0.002213	1.0	0.1	0.0024	92.4	1	1	0.01
n=10, m=10	5.0	2	0.133279	1.0	0.1	0.1	4.8	4	4	0.01
n=10, m=10	5.0	28	0.002224	1.0	0.1	0.00238	97.2	1	1	0.01
n=10, m=10	5.0	5	0.001348	1.0	0.1	0.00108	64.2	1	1	0
n=10, m=10	5.0	1	0.197933	1.0	0.1	0.1	54.6	4	4	0
n=10, m=10	5.0	5	0.001000	1.0	0.1	0.0009547	27.0	1	1	0
r=10, m=10	5.0	1	0.127809	1.0	0.1	0.1	3.6	4	4	0
n=20, m=20	6.0	16	0.172247	0.5	0.05	0.05	54.0	8	8	0.01
n=20, m=20	6.0	2	0.201409	2.0	0.1	0.1	60.0	4	4	0.001
n=20, m=20	6.0	1	0.196550	1.0	0.1	0.1	60.	4	4	0
n=20, m=20	6.0	2	0.185085	0.5	0.05	0.05	121.8	4	4	0.001
n=30, m=30	2.04	1	1.189703	1.0	0.1	0.1	322.2	4	4	0
n=30, m=30	2.04	2	1.483190	2.0	0.1	0.1	283.2	5	5	0.001
n=30, m=30	2.04	2	0.549478	0.1	0.05	0.05	669	5	5	0.001
n=30, m=30	2.04	2	1.914232	1.0	0.1	0.1	330	5	5	0.001
n=30, m=30	2.04	2	0.802402	0.1	0.05	0.05	553	10	10	0.01

MAIN PROGRAM


INPUT
n, m, M, h, ℓ, ε, τ,
(a_{ij}), (b_i) j = 1(1)n i = 1(1)m
(y_{0j}) (c_j) j = 1(1)n
$ny_i = b_i - \sum_{j=1}^{n} a_{ij} y_{0j}$ i = 1(1)m
using subroutine INTLZE evaluate the first vector x in the direction normal to the cost plane through y_0
save first vector in family
$f_{1j} = x_j$ j = 1(1)n
$fc_1 = x \cdot c$ cost of x
k := 1 sets loop counter
do buckshot and asexual improvement in subroutine FMLY with the given first vector: f_{1j} j = 1(1)n
rank vectors by cost and find next best vector x' and store in f_1.
$t := \sqrt{\sum_{j=1}^{n} (x'_j - x_j)^2}$
k := k+1
t < ε ?
YES
NO
PRINT OUTPUT AND STOP
k ≤ M ?
NO
YES
PRINT INTERMEDIATE OUTPUT
ℓ < t ?
NO
YES
ℓ := ½ ℓ

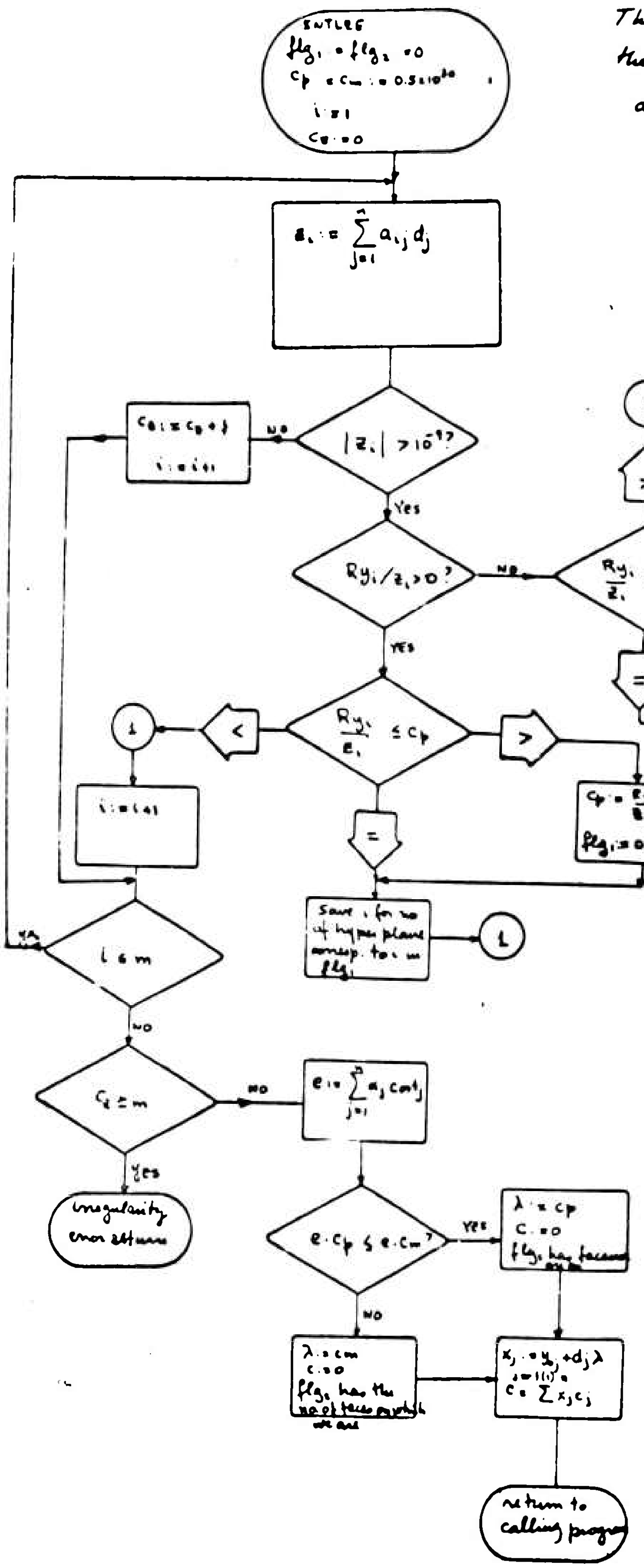
n	corresponds to the dimension of the system
m	= the number of constraint planes
M	= the maximum number of loops
h	= the high cons factor
ℓ	= the low cons factor
ε	exit tolerance for successive approximations
τ	asexual improvement tolerance
(a_{ij}), (b_i)	constraint matrix
(y_{0j})	initial feasible vector
(c_j)	cost function

SUBROUTINE FMLY


given running bid vector x and initial feasible vector y0, T we inform the annual change and the buckshot effect in this routine
$d_j := y_{0j} - x_j \quad j=1(1)n$
$h_f := h \times$ random number between 0 and 1
L:=1
$d_L := d_L \pm l_1 h_f$
go to INTLZE to get x^1, x^2, x^3, x^4 and correspon-ding costs c_1, c_2, c_3, c_4
$c_k + T < c$?
YES
$x_j := x_j^k \quad j=1(1)n$
$c := c_k$
NO
save in family:
$f_{4L-k+1,j} := x_j^k \quad j=1(1)n \quad k=1(1)4$
$f_{4L-k+1} := c_k$
L := L+1
L ≤ n ?
YES
NO
$x'_j := x_j \quad j=1(1)n$ the running bid
L := 1
K := L+1
is f_L on the same face as f_K ?
NO
YES
$d_j := y_{0j} - (f_{Lj} + f_{Kj})\frac{1}{2}$
j=1(1)n
go to ENTLZE to find the vector on the face of the polyhedron in the direction d through y_0: x', c'
c ≤ c' ?
YES
NO
Exchange c and c' and x and x'
K := K + Kstep, to skip over same faces
K ≤ 4n+1 ?
YES
NO
L := L + Lstep, skip over same faces to Lstep
L ≤ 4n ?
YES
NO
RETURN TO CALLING PROGRAM

SUBROUTINE SNTLZE

This subroutine finds the point on the line through y_0 with direction d on the constraint hyperplane.


SNTLZE
$flg_1 := flg_2 := 0$
$c_p := c_m := 0.5\times10^{10}$
$i := 1$
$c_0 := 0$
$z_i := \sum_{j=1}^{n} a_{ij} d_j$
$|z_i| > 10^{-9}$?
NO
$c_0 := c_0 + 1$
$i := i+1$
YES
$Ry_i/z_i > 0$?
NO
$\frac{Ry_i}{z_i} \geq c_m$?
$c_m := \frac{Ry_i}{z_i}$
$flg_2 := 0$
save i for no. of hyperp. corresp to i in flg_2
YES
$\frac{Ry_i}{z_i} \leq c_p$
$c_p := \frac{Ry_i}{z_i}$
$flg_1 := 0$
$i := i+1$
Save i for no of hyper plane corresp. to i in flg_1
1
<
>
=
YES
$i \leq m$
NO
$c_0 = m$
Yes
irregularity error return
NO
$e := \sum_{j=1}^{n} a_j c_m t_j$
$e \cdot c_p \leq e \cdot c_m$?
YES
$\lambda := c_p$
$c := 0$
flg_1 has faces on line
NO
$\lambda := c_m$
$c := 0$
flg_2 has the no of faces on which we are
$x_j := y_j + d_j \lambda$
$j = 1(1)n$
$c = \sum x_j c_j$
return to calling program

CONVEX PROGRAMMING

The method we discussed in the linear programming part was extended in a "natural" way to try to solve convex programming problems. By convex programming we mean: given a convex set (whose constraints or boundaries are not necessarily defined by linear functions), and a linear* cost function, we want to find a point in the convex set such that no other point in it has a smaller cost. If the constraints are linear the problem reduces to linear programming. We note that the linearity of the constraints in the linear programming method was used exclusively for finding the intersection of a given line with the boundary, and nowhere else. Thus this method becomes suitable for generalization as soon as we have a way of finding the intersection point of a line with the boundary of the convex set.

Let us now analyze the method that we used to find the point on the boundary given by the intersection of the constraint surface and a line defined by a vector of direction numbers and a point through which the line must go:

The equation of a line through a point y with given direction numbers $\vec{d} = (d_1, \cdots, d_n)$ is given by
$\vec{x} = (x_1, \cdots, x_n) = (y_1 + td_1,\ y_2 + td_2, \cdots, y_n + td_n)$ or simply
$\vec{x} = \bar{y} + t\vec{d}$.

If we define our convex set as the intersection of several convex sets given by the following equations:

$\varphi_i(\bar{x}) \leq b_i$, $i = 1, \cdots, m$, where $\varphi(x)$ are continuous functions.

Then we can simply say that we want to find a λ such that:

* Note: The general convex programming case admits a convex objective function.

$\varphi_i(\vec{y} + \lambda\vec{d}) \leq b_i$ for $i = 1, \cdots, m$ and for some j

$\varphi_j(\vec{y} + \lambda\vec{d}) = b_i$.

Since the convex sets are bounded we can say that for a given $t > 1$ we can find an n such that

$\varphi_j(\vec{y} + t^n\vec{d}) > b_j$ for some j and

$\varphi_i(\vec{y} + t^{n-1}\vec{d}) \leq b_i$ for $i = 1, \cdots, m$ since y is inside the convex set. We now proceed by fixing $|t| > 1$, then we can find n by multiplying t by $|t|$ until we find the desired n. We then know that $t^{n-1} \leq \lambda < t^n$, the order may be reversed if $t < 0$. We can then by trial and error find λ by a binary search within the interval. If we let $\tau > 0$ denote a tolerance which we fixed before we start our search for λ we can continue as follows: For notational simplicity let us assume $t > 0$.

Let $\alpha = t^n - t^{n-1}$ and $\beta = t^{n-1}$. If $\varphi_i(\vec{y} + (\beta + \frac{1}{2}\alpha)\vec{d}) \leq b_i$ for $i = 1, \cdots, m$, and for some j, $b_i - \varphi_j(\vec{y} + (\beta + \frac{1}{2}\alpha)\vec{d}) < \tau$ then we define $\lambda = \beta + \frac{1}{2}\alpha$. If for $i = 1, \cdots, m$ $b_i - \varphi_i(\vec{y} + (\beta + \frac{1}{2}\alpha)\vec{d}) > \tau$ then let us replace $\beta + \frac{1}{2}\alpha$ for β and $\frac{1}{2}\alpha$ for α and continue. If $\varphi_j(\vec{y} + (\beta + \frac{1}{2}\alpha)\vec{d}) > b_j$ then we replace $\beta - \frac{1}{2}\alpha$ for β and $\frac{1}{2}\alpha$ for α and continue. This method converges because the constraint surfaces are continuous. In this manner we find the point $\vec{x}$ which lies on the line with direction numbers $\vec{d}$ through y within τ of at least one constraint surface.

Since in our method we are searching within a small cone of a given best vector we know that subsequent λ's do not differ by very much, in general, and are good approximations to each other. However, even if the approximations were not good we have a convergence for starting $|\lambda| > 1$ at the beginning of each cycle, and this is easily guaranteed. For a precise description of the method we refer the reader to the flow chart at the end of this section, p. 51.

A CLASS OF CONVEX PROGRAMMING PROBLEMS

As convex sets we used the n-dimensional ellipsoids defined by

$$\varphi_i(x_1,\cdots,x_n) = \sum_{j=1}^{n} \left(\frac{x_j-a_{ij}}{b_{ij}}\right)^2 \leq e_i, \quad \text{where} \quad i = 1, \cdots, m,$$

and $m, n \leq 10$.

This family was chosen for expediency reasons since any such function is easily evaluated and therefore fast on the computer. The answers are also readily checked for the case $m = 1$, as will be established in the sequel. For $m > 1$ the problem is more complicated because the answer may lie on the curve defined by the intersection of the m ellipsoids. However, the main reason for using $m = 1$ in our experiments was that m is a factor of the length of time the computer takes in doing a particular example. This can be readily verified since we spend most of our machine time in finding the intersection of a line and the surface bounding the set, and there are m of these equations to evaluate each time. Furthermore, it seems reasonable to assume that if the method can solve problems for $m = 1$ then it can also solve them for $m > 1$.

We are given together with the ellipsoid

$\sum_{j=1}^{n} \left(\frac{x_j-a_{ij}}{b_{ij}}\right)^2 \leq e_i$ a linear cost function

$\vec{c} \cdot \vec{x} = \sum_{j=1}^{n} c_j x_j$ which we are to minimize. The answer to the problem then is: for $m = 1$, the point of tangency at the ellipsoid such that the tangent hyperplane has the direction numbers given by $\vec{c} = (c_1, \cdots, _n)$, and such that the $\sum_{j=1}^{n} c_j x_j$ is the smaller of the two possible outcomes.

Without loss of generalization we can assume that $\sum_{j=1}^{n} (\frac{x_j - a_j}{b_j}) \leq e$ can be reduced to $\Sigma(\frac{x_m}{b_j})^2 \leq e$, by simply translating the ellipsoid. Since we are interested only in the boundary point we have

$$\sum_{j=1}^{n} (\frac{x_j}{b_j})^2 = e.$$

From elementary calculus we have that the tangent plane at a point z can be represented by: $\sum_{j=1}^{n} \frac{x_j z_j}{b_j^2} = e$. And this plane is to be parallel to $\sum_{j=1}^{n} c_j x_j$, consequently we have $(\frac{z_1}{b_1^2}, \frac{z_2}{b_2^2}, \cdots, \frac{z_n}{b_n^2}) = \lambda(c_1, c_2, \cdots, c_n)$ that is the direction numbers are proportional. Hence we have $z_i = \lambda\, b_i^2\, c_i$. Substituting in the equation for the ellipse we have:

$$\sum_{i=1}^{n} \frac{\lambda^2 b_i^2 c_i^2}{b_i^2} = e \quad \therefore \quad \lambda^2 = e/(\sum_{i=1}^{n} b_i^2\, c_i^2) \quad \therefore \quad \text{and} \quad \lambda = -\sqrt{e/(\sum_{i=1}^{n} b_i^2 c_i^2)}.$$

Since for our experimentation we chose $\vec{c}$ such that $c_i \geq 0$ and at least one $c_j > 0$, we could always solve for two real distinct λ's. From these we can always pick the λ which gives us the smallest cost when we substitute for the answer. If we denote by $\vec{a}$ the center of the ellipse then the answer is $\vec{x} = \vec{a} + \lambda(b_1^2 c_1, b_2^2 c_2, \cdots, b_n^2 c_n)$. Note that this expression is readily computable.

Another important aspect that aided us in the choice of these functions was the ease with which one can make changes in the format of the ellipsoid by simply changing the coefficients. We can make the convex set be an elongated "cigar" in which the search would be more difficult, if we make the ratio of length to width large enough. This problem was suggested by Bremermann and Salaff [7].

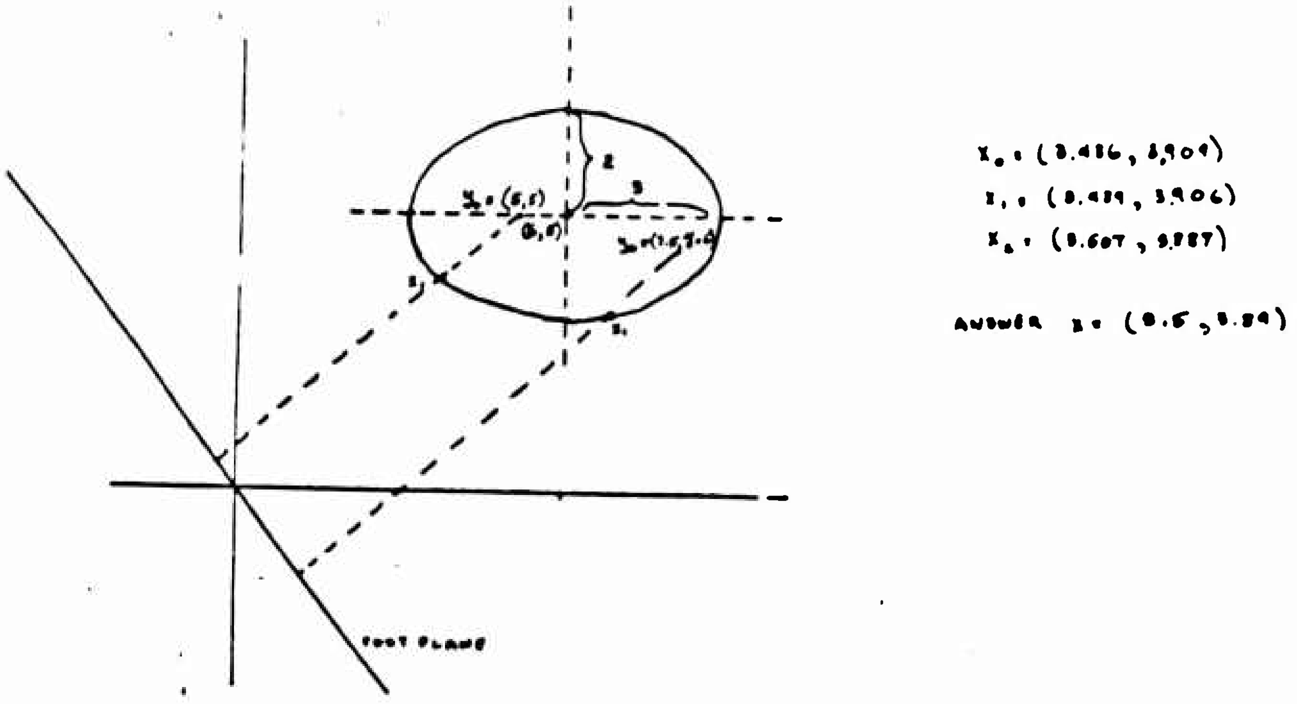
The program that we ran on the IBM 7094 was written in Fortran and FAP. Its structure is in general the same as the modified version of the linear programming problems. That is to say we built into the program the capacities we found advantageous in our previous experiments. The program allows "asexual" improvements during the " arent" generation stage. For a detailed view of the process we refer the reader to the flow charts at the end of this section, pp. 49 to 51.

Most of the experiments we ran were with ellipsoids whose coefficients were rather well behaved, that is, in a range between 1 and 10. The experiments were geared mostly to find out how the method behaves as we let n grow from 2 to 10.

We begin by analyzing one of the simplest experiments possible:

$$\left(\frac{x-6}{3}\right)^2 + \left(\frac{y-5}{2}\right)^2 \leq 1$$

with a cost function $z = x + y$ to be minimized. The point of departure was $y_0 = (5,5)$ which is within the ellipse



In this case we can see that given a very propicious start the progress is as one would expect it: direct and fast. This particular run took 2 loops and 1.2 seconds. Most of it, however, is spent on the input and output of information. We note that x comes to within 2.57×10^{-7} of the ellipse. If we now do the same problem but start with a y_0 of (7.5, 4.5), our progress is a bit slower , the sequence of best vectors being:

$$x_0 = (5.533,\ 3.024),\ x_1 = (4.495,\ 3.27),\ x_3 = (3.504,\ 3.891).$$

Between these vectors a great many more asexual improvements come to span the x_0, x_1, x_2. For the former case we only have 2 asexual improvements; for the latter we have 20. It is noteworthy that the latter also took 2.4 seconds to converge to approximately the same answer.

Let us now consider a case which is an "elongated cigar shaped" ellipse:

$$\left(\frac{x-10}{10}\right)^2 + \left(\frac{y-0.1}{0.1}\right)^2 \leq 1.$$

This defines an ellipse of length 100 units and width of 1. The starting point we took was (19,0.09). For a cost function of $c = (10,1)$ the sequence of test vectors was:

$x_0 = (17.85, 0.03806)$, $x_1 = (0.1712, 0.08157)$, $x_2 = (0.1611, 0.08$

$x_3 = (0.01561, 0.09442)$, $x_4 = (0.01486, 0.09455)$.

The answer to the problem is $(0.1 \times 10^{-6}, 0.09999)$. This run took 7.2 seconds. When we changed the large cone factor from 1.2 to 0.8 our method fared much better. For in the sixth loop the answer was $x_6 = (0.42 \times 10^{-5}, 0.09995)$. This result is in line with our previous experiments in linear programming, where we found that a smaller cone factor led in general to a better approximation at the expense of a greater number of iterations. The latter case took 6 seconds. In a subsequent run we changed the cost function to make for a less well defined point of tangency. We let $c = (1,10)$. The method converges as follows:

$x_0 = (18.32, 0.04452)$, $x_1 = (.1849, 0.08086)$, $x_2 = (.1779, 0.08122)$.

The answer is $x = (0.0458, 0.0005)$. At this point we should note that the method does not behave very well when the tangent to the curve makes a very small angle with the line at the intersection defining the points on the curve. This is due to the constants used in the convergence to the point of intersection and also partly due to the limitations of the floating point arithmetic of the computer. Although this problem is surmountable we did not endeavor to conquer it, the interest being marginal.

Let us now investigate a typical 3-dimensional experiment $(\frac{x-6}{3})^2 + (\frac{y-5}{2})^2 + (\frac{z-5}{4})^2 \leq 1$ with a cost function $c = (1,1,1)$, a cone factor of 1.2 and an initial feasible vector $y_0 = (5,5,4)$. To iterate twice the method took 6 seconds and converged to $x_2 = (4.339, 4.262, 2.014)$, the correct answer being $x = (4.31, 4.255, 2.02)$. When we changed the cost plane to $c = (1,2,3)$ we converged in 7.2 seconds to $x_3 = (5.318, 4.373, 1.312)$, $x = (5.29, 4.368, 1.21)$.

If we took a long cigar in three dimensions we faced the same sort of problem that we had in the 2-dimensional case. For example let us look at the following:

$(\frac{x-20}{20})^2 + (\frac{y-0.1}{0.1})^2 + (\frac{z-0.1}{0.1})^2 \leq 1$ with a cost plane

$c = (10,1,1)$ and an initial feasible vector $y_0 = (15, 0.09, 0.09)$. For this case we stopped after 58.8 seconds at $x_{11} = (0.0008714, 0.1003, 0.09911)$. The true answer is $x = (0.1\times10^{-7}, 0.09995, 0.09995$

Let us now analyze a six dimensional example:

$$\left(\frac{x_1-4}{2}\right)^2 + \left(\frac{x_2-4}{3}\right)^2 + \left(\frac{x_3-4}{4}\right)^2 + \left(\frac{x_4-4}{4}\right)^2 + \left(\frac{x_5-4}{3}\right)^2 + \left(\frac{x_6-4}{2}\right)^2 \leq 1$$

with a cost function $c = (1,1,1,1,1,1)$ and $y_0 = (4,4,4,4,4,4)$. In 16.2 seconds the program iterated twice and converged to $x_2 = (3.453,\ 2.818,\ 1.962,\ 1.912,\ 2.799,\ 3.444)$ the true answer being: $x = (3.475,\ 2.82,\ 1.89,\ 1.89,\ 2.82,\ 3.475)$. This experiment was relatively trivial since we started out from the center of the ellipse. However, when we tried the following 10-dimensional case:

$$\left(\frac{x_1-4}{3}\right)^2 + \left(\frac{x_2-3}{2}\right)^2 + \left(\frac{x_3-4}{4}\right)^2 + \left(\frac{x_4-5}{3}\right)^2 + \left(\frac{x_5-2}{3}\right)^2 + \left(\frac{x_6-2}{3}\right)^2$$

$$\left(\frac{x_7-4}{3}\right)^2 + \left(\frac{x_8-3}{2}\right)^2 + \left(\frac{x_9-4}{4}\right)^2 + \left(\frac{x_{10}-2}{4}\right)^2 \leq 1$$ with a cost

function $c = (1,2,2,3,2,4,1,2,2,3)$ and

$y_0 = (4.3,\ 3.2,\ 4.4,\ 5.3,\ 3.3,\ 2.3,\ 4.3,\ 3.2,\ 4.4,\ 2.4)$.

The method iterated 5 times in 91.2 seconds and stopped at:

$x_5 = (3.598,2.664,2.698,3.931,2.297,0.4893,3.631,2.672,$
$2.675,\ -0.04223)$ compared to the true solution

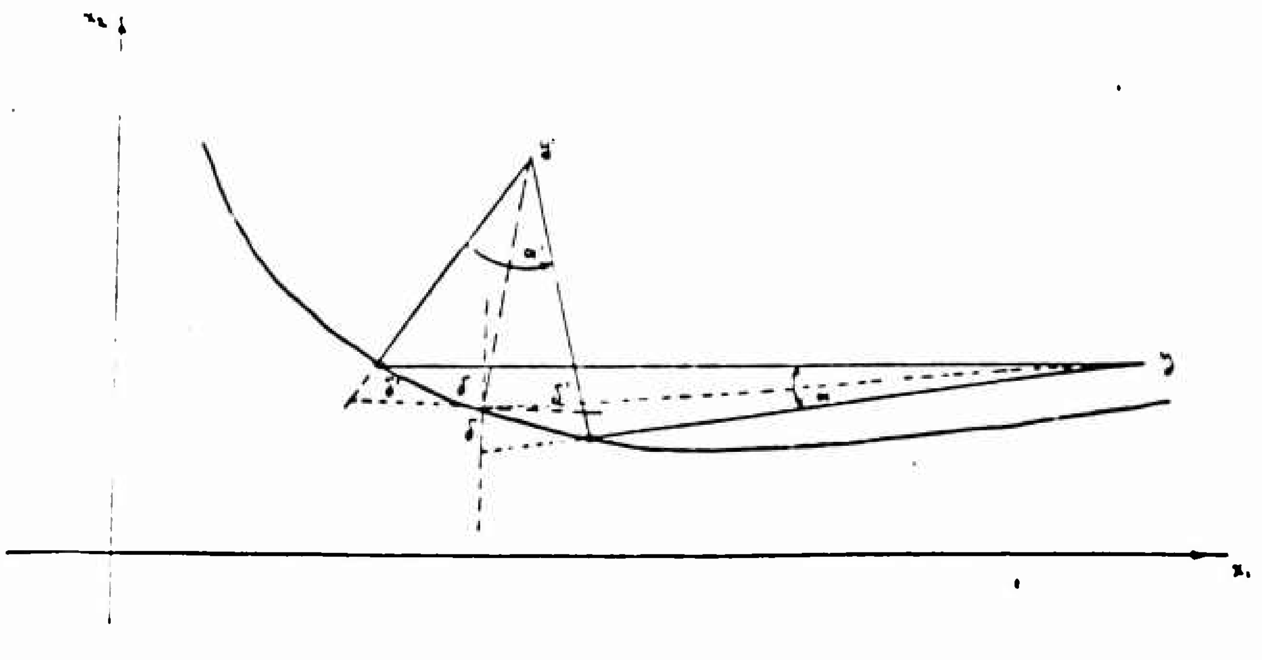
$x = (3.627,2.669,2.675,3.882,2.255,0.507,3.627,2.669,2.675,0.01)$

Before we summarize our results let us discuss one two-dimensional intersection of two ellipses which we ran.

$$\begin{cases} \left(\frac{x-5}{4}\right)^2 + \left(\frac{y-5}{2}\right)^2 \leq 1 \\ \left(\frac{x-5}{2}\right)^2 + \left(\frac{y-5}{4}\right)^2 \leq 1 \end{cases}$$

With a cost function $c = (1,1)$ and a start $y_o = (6.5,5)$ in two loops it converged to $(3.209, 3.222)$. We note that this point is within 0.0936 from the boundary of the first ellipse and 0.0077 from the second one. That is to say, it is approximately at the intersection where it should be.

In view of our experiments we find that what would be expected from the consequences of the more widely experimented linear programming would also hold true in the convex programming case. Namely, we would expect that the smaller the cone for the "buckshot" the better the approximation and the lengthier the process. This leads us to believe that the method has not been tested widely enough, and, even where tested, not all the features that the experiments suggest were implemented. For example, if one wants to develop an efficient algorithm based on this approach on should investigate the possibility of a "self organizing" cone. That is, the dimensions of the cone would be based on some possibly gross estimation of what the surface "looks" like in the neighborhood of a given best point. This suggests itself if we consider the following case



If we make even a small change in coordinates at x and draw a line $\vec{x} \pm \delta \vec{e}_1$ then we see the angle α is small but the effect is the same as for a much larger δ' from a different point y'. This also points to the need of moving the initial feasible vector to more favorable spots in the convex set.

In spite of some shortcomings of the program, some results are very encouraging. Even the poor convergence in ellipsoids whose ratio of largest to smallest diameter is large we find encouragement, for if we suitably modify the basic feasible vector we think that the method can become more accurate.

If by convex set we understand a set such that given two points in it the line segment joining them is also contained by the set, then we want to note that in our discussion we used this property only very superficially. What we used most was our assumption that the given sets were bounded, and this only in the direction toward the cost plane, and closed, that is we could find the point on the boundary. From this we thought that another possible application would be to use a modified technique for finding roots of polynomials, around a local "guess", in the complex plane. This seems a feasible proposition since polynomials are easy to evaluate on the computer. If the constraint functions are difficult to evaluate or take a long time to be computed, then the method might break down from the point of view of time elapsed to find the answer, for the main part the program loops in evaluating the constraint functions. The following table has a summary of all our experiments.

/

TABLE 3 (Page 1)

Dimension	Number of Iterations	Largest Error	Large Cone Factor	Small Cone Factor	Asexual Acceptance	Ellipsoid Equation	Initial Feasible Vector	Cost Plane	Time in Seconds
n=2, m=1	1	0	1.2	0.5	0.01	$(\frac{x_1-6}{3})^2 + (\frac{x_2-3}{2})^2 \leq 1$	(5,5)	(1,0)	1.2
n=2, m=1	2	0.007	1.2	0.5	0.01	$(\frac{x_1-6}{3})^2 + (\frac{x_2-3}{[illegible]})^2 \leq 1$	(5,5)	(1,1)	2.4
n=2, m=1	3	.004	1.2	0.5	0.01	$(\frac{x_1-6}{3})^2 + (\frac{x_2-3}{5})^2 \leq 1$	(7.5,4.5)	(1,1)	2.4
n=2, m=1	4	0.01486	1.2	0.5	0.01	$(\frac{x_1-10}{10})^2 + (\frac{x_2-0.1}{0.1})^2 \leq 1$	(19,0.09)	(10,1)	3.6
n=2, m=1	4	0.00003	0.8	0.03	0.01	$(\frac{x_1-10}{10})^2 + (\frac{x_2-0.1}{0.1})^2 \leq 1$	(19,0.09)	(10,1)	3.0
n=2, m=1	1	0.05013	1.2	0.5	0.01	$(\frac{x_1-10}{10})^2 + (\frac{x_2-0.1}{0.1})^2 \leq 1$	(19,0.09)	(20,0)	0.[illegible]
n=2, m=1	2	0.006540	1.2	0.5	0.01	$(\frac{x_1-20}{20})^2 + (\frac{x_2-0.1}{0.1})^2 \leq 1$	(29,0.09)	(10,1)	7.2
n=2, m=1	2	0.1379	1.2	0.5	0.01	$(\frac{x_1-10}{10})^2 + (\frac{x_2-0.1}{0.1})^2 \leq 1$	(19,0.09)	(1,10)	?
n=2, m=1	3	0.00521	0.8	0.08	0.01	$(\frac{x_1-10}{10})^2 + (\frac{x_2-0.1}{0.1})^2 \leq 1$	(19,0.09)	(1,10)	2.25
n=2, m=1	2	0.0105	1.2	0.5	0.01	$(\frac{x_1-10}{10})^2 + (\frac{x_2-0.1}{0.1})^2 \leq 1$	(19,.09)	(1,20)	1.58
n=2, m=1	8	0.0741	1.2	0.5	0.01	$(\frac{x_1-20}{20})^2) + (\frac{x_2-0.1}{0.1})^2 \leq 1$	(29,.09)	(1,10)	6.8

TABLE 3 (Page 2)

Dimension	Number of Iterations	Largest Error	Large Cone Factor	Small Cone Factor	Asexual Acceptance	Ellipsoid Equation	Initial Feasible Vector	Cost Plane	Time in Seconds
n=3, m=1	3	0	1.2	0.5	0.01	$(\frac{x_1-6}{3})^2 + (\frac{x_2-5}{2})^2$	(5,5,4)	(1,0,0)	1.8
						$+ (\frac{x_3-5}{4})^2 \leq 1$			
n=3, m=1	2	0.042	1.2	0.5	0.01	$(\frac{x_1-6}{3})^2 + (\frac{x_2-5}{2})^2$	(5,5,4)	(0,1,1)	3.0
						$+ (\frac{x_3-5}{4})^2 \leq 1$			
n=3, m=1	2	0.010	1.2	0.5	0.01	$(\frac{x_1-6}{3})^2 + (\frac{x_2-5}{2})^2$	(5.5, 6.5,6)	(1,1,0)	3.6
						$+ (\frac{x_3-5}{4})^2 \leq 1$			
n=3, m=1	2	0.009	1.2	0.5	0.01	$(\frac{x_1-6}{3})^2 + (\frac{x_2-5}{2})^2$	(5,5,4)	(1,1,1)	3.0
						$+ (\frac{x_3-5}{4})^2 \leq 1$			
n=3, m=1	3	0.012	1.2	0.5	0.01	$(\frac{x_1-6}{3})^2 + (\frac{x_2-5}{2})^2$	(5,5,4)	(1,2,3)	3.6
						$+ (\frac{x_3-5}{4})^2 \leq 1$			
n=3, m=1	2	0.011	1.2	0.5	0.01	$(\frac{x_1}{1})^2 + (\frac{x_2-4}{2})^2$	(0.5, 5,5)	(0,1,1)	2.4
						$+ (\frac{x_3-5}{3})^2 \leq 1$			

TABLE 3 (Page 3)

Dimension	Number of Iterations	Largest Error	Large Cone Factor	Small Cone Factor	Asexual Acceptance	Ellipsoid Equation	Initial Feasible Vector	Cost Plane	Time in Seconds
n=3, m=1	3	0.030	1.2	0.5	0.01	$(\frac{x_1}{1})^2 + (\frac{x_2-4}{2})^2$	(0.5, 5,5)	(1,1,1)	4.8
						$+ (\frac{x_3-5}{3})^2 \leq 1$			
n=3, m=1	4	0.021	1.2	0.12	0.01	$(\frac{x_1-20}{20})^2 + (\frac{x_2-0.1}{0.1})^2 +$	(15,0.09, 0.09)	(1,10, 10)	5.006
						$(\frac{x_2-0.1}{0.1})^2 \leq 1$			
n=3, m=1	11	8×10^{-3}	1.2	0.12	0.01	$(\frac{x_1-20}{20})^2 + (\frac{x_2-0.1}{0.1})^2 +$	(15,0.09, 0.09)	(10,1, 1)	28.8
						$(\frac{x_3-0.1}{0.1})^2 \leq 1$			
n=3, m=1	15	1×10^{-3}	1.2	0.12	0.01	$(\frac{x_1-10}{10})^2 + (\frac{x_2-0.1}{0.1})^2 +$	(15,0.09, 0.09)	(10,1, 1)	52.8
						$(\frac{x_3-0.1}{0.1})^2 \leq 1$			
n=4, m=1	29	0.001848	0.4	0.04	0.01	$(\frac{x_1-10}{10})^2 + (\frac{x_2-0.1}{0.1})^2 +$	(15,0.09, 0.09,0.09)	(10,1, 1,1)	75.6
						$(\frac{x_3-0.1}{0.1})^2 + (\frac{x_4-0.1}{0.1})^2 \leq 1$			

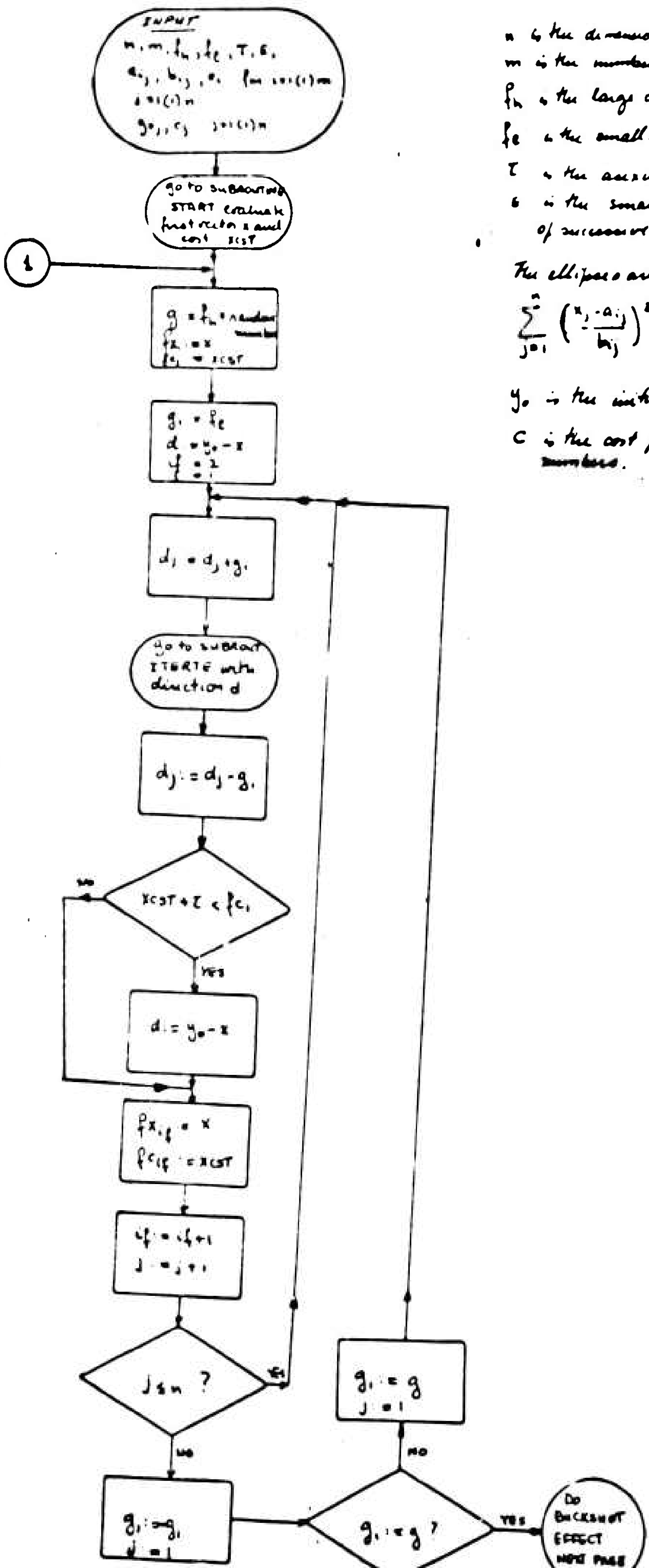
TABLE 3 (Page 4)

Dimension	Number of Iterations	Largest Error	Large Cone Factor	Small Cone Factor	Asexual Acceptance	Ellipsoid Equation	Initial Feasible Vector	Cost Plane	Time in Seconds
n=6, m=1	1	0.	1.2	0.5	0.01	$(\frac{x_1-4}{2})^2 + (\frac{x_2-4}{[illegible]})^2$	(4,4,4, 4,4,4)	(1,0,0, 0,0.0)	3.6
						$+ (\frac{x_3-4}{2})^2 + (\frac{x_4-4}{4})^2$			
						$+(\frac{x_5-4}{3})^2 + (\frac{x_6-4}{2}) \leq 1$			
n=6, m=1	2	0.15	1.2	0.5	0.01	$(\frac{x_1-4}{2})^2 + (\frac{x_2-4}{3})^2$	(4,4,4, 4,4,4)	(1,1,0, 0,0,0)	10.2
						$+ (\frac{x_3-4}{2})^2 + (\frac{x_4-4}{4})^2$			
.						$+(\frac{x_5-4}{3})^2 + (\frac{x_6-4}{2}) \leq 1$			
n=6, m=1	3	0.024	1.2	0.5	0.01	$(\frac{x_1-4}{2})^2 + (\frac{x_2-4}{3})^2$	(4,4,4 4,4,4)	(1,1,1, 0,0,0)	16.2
						$+ (\frac{x_3-4}{2})^2 + (\frac{x_4-4}{4})^2$			
						$+(\frac{x_5-4}{3})^2 + (\frac{x_6-4}{2}) \leq 1$			
n=6, m=1	3	0.12	1.2	0.5	0.01	$(\frac{x_1-4}{2})^2 + (\frac{x_2-4}{3})^2$	(4,4,4 4,4,4)	(1,1,1, 1,0,0)	15.0
						$+ (\frac{x_3-4}{2})^2 + (\frac{x_4-4}{4})^2$			
						$+(\frac{x_5-4}{3})^2 + (\frac{x_5-4}{2}) \leq 1$			

TABLE 3 (Page 5)

Dimension	Number of Iterations	Largest Error	Large Cone Factor	Small Cone Factor	Asexual Acceptance	Ellipsoid Equation	Initial Feasible Vector	Cost Plane	Time in Seconds
n=6, m	3	0.09	1.2	0.5	0.01	$(\frac{x_1-4}{2})^2 + (\frac{x_2-4}{3})^2$	(4,4,4, 4,4,4)	(1,1,1, 1.1,0)	16.2
						$+ (\frac{x_3-4}{2})^2 + (\frac{x_4-4}{4})^2$			
						$+ (\frac{x_5-4}{3})^2 + (\frac{x_6-4}{2}) \leq 1$			
n=6, m=1	3	0.062	1.2	0.5	0.01	$(\frac{x_1-4}{2})^2 + (\frac{x_2-4}{3})^2$	(4,4,4, 4,4,4)	(1,1,1, 1,1,1)	16.2
						$+ (\frac{x_3-4}{2})^2 + (\frac{x_4-4}{4})^2$			
						$+ (\frac{x_5-4}{3})^2 + (\frac{x_6-4}{2}) \leq 1$			
n=10, m=1	5	0.05223	1.2	0.5	0.01	$(\frac{x_1-4}{3})^2 + (\frac{x_2-3}{2})^2$	(4.3,3.2, 4.4,5.3, 3.3,2.3, 4.3,3.2, 4.4,2.4)	(1,2,2, 3,2,4, 1,2,2, 3)	93.
						$+ (\frac{x_3-4}{4})^2 + (\frac{x_4-5}{3})^2$			
						$+ (\frac{x_5-2}{3})^2 + (\frac{x_6-2}{3})^2$			
						$+ (\frac{x_7-4}{3})^2 + (\frac{x_8-3}{2})^2$			
						$+ (\frac{x_9-4}{4})^2 + (\frac{x_{10}-2}{4})^2 \leq 1$			
n=2, m=2	2	0.0011	1.2	0.5	0.01	$(\frac{x_1-5}{4})^2 + (\frac{x_2-5}{2})^2 \leq 1$	(6.5,5)	(1,1)	not avail.
						$(\frac{x_1-5}{2})^2 + (\frac{x_2-5}{[illegible]})^2 \leq 1$			

MAIN PROGRAM

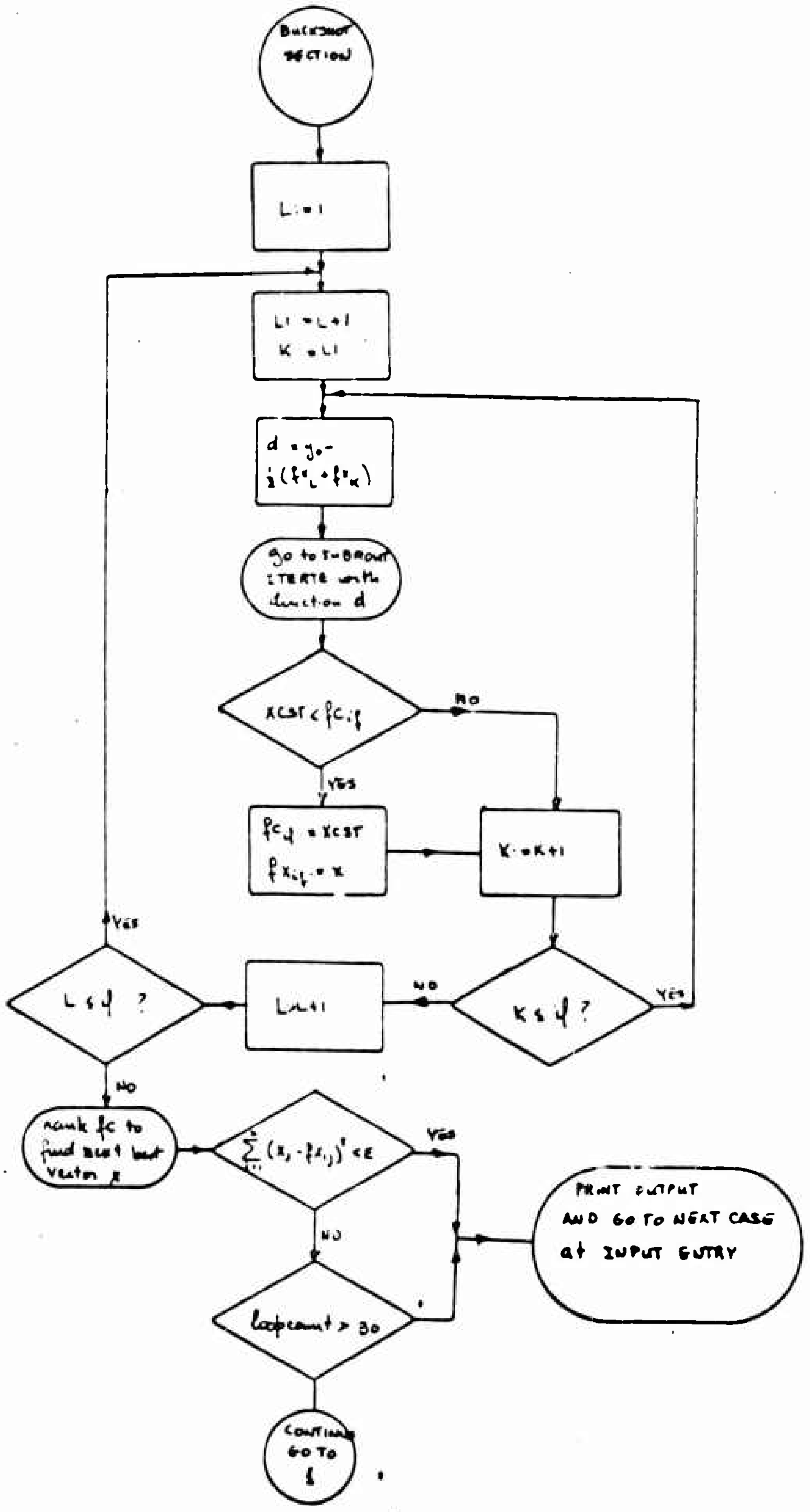


n is the dimension of the space
m is the number of intersecting ellipses
f_m is the large cone factor
f_e is the small cone factor
τ is the accrual tolerance.
ϵ is the smallest acceptable change of successive "best vectors"

The ellipses are given by:

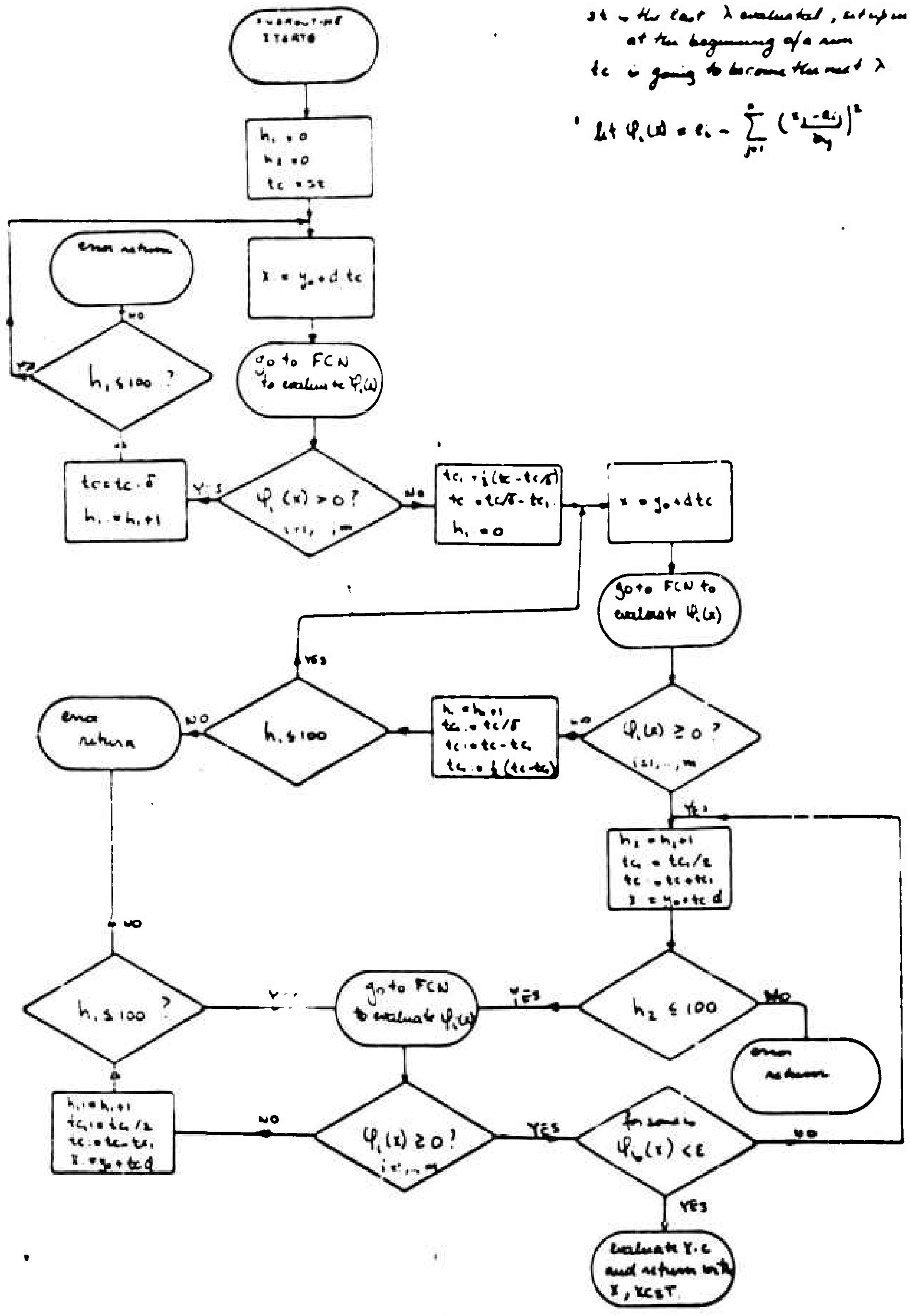
$$\sum_{j=1}^{n} \left(\frac{x_j - a_{ij}}{b_{ij}}\right)^2 \le e_i \quad \text{for } i=1,\dots,m$$

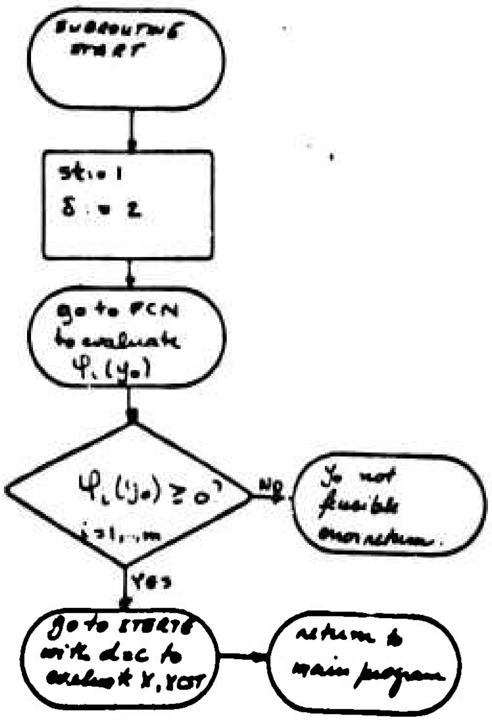
y_0 is the initial feasible vector.
C is the cost plane's direction numbers.

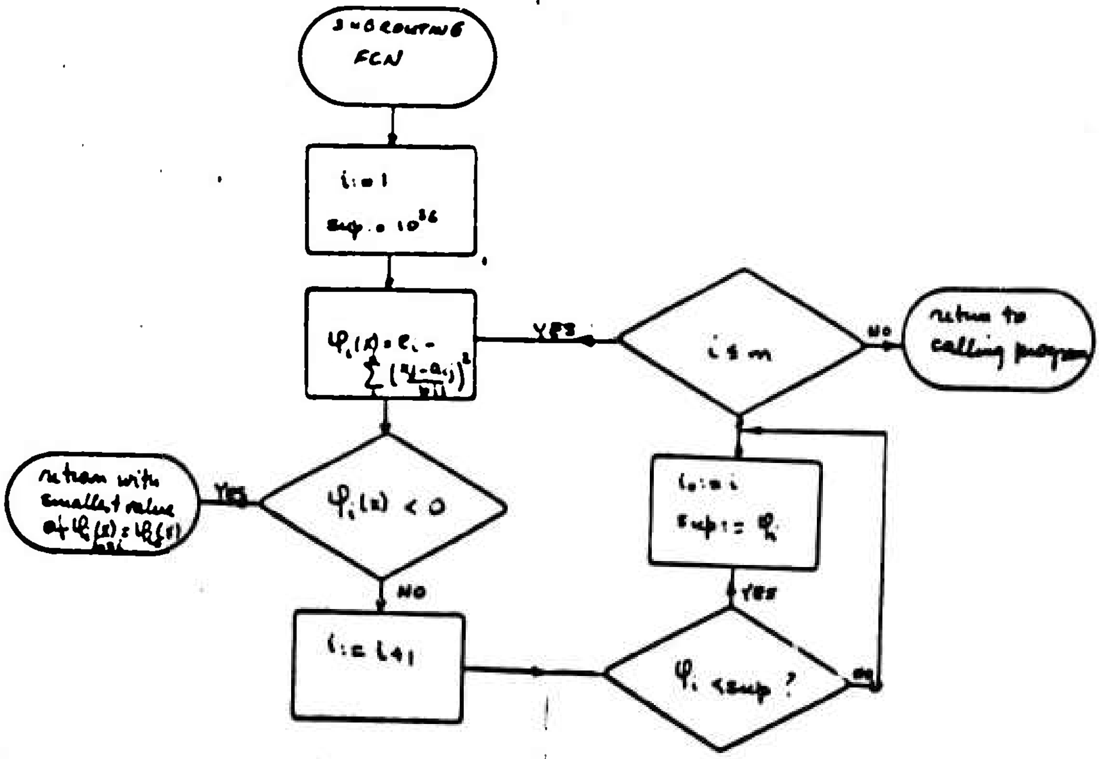

BUCKSHOT SECTION
L = 1
L1 = L+1
K = L1
d = y0 - ½(fx_L + fx_K)
go to SUBROUT ITERTE with direction d
XCST < fc_iq
NO
YES
fc_iq = XCST
fx_iq = x
K = K+1
K ≤ q ?
YES
NO
L = L+1
L ≤ q ?
YES
NO
rank fc to find best vector x
Σ (x_i - fx_i,j)² < ε
YES
NO
loopcount > 30
PRINT OUTPUT AND GO TO NEXT CASE at INPUT ENTRY
CONTINUE GO TO [illegible]

st = the last λ evaluated, set up in START
at the beginning of a run
tc is going to become the next λ

let $\varphi_i(x) = c_i - \sum_{j=1}^{n} \left(\frac{x_j - a_{ij}}{b_{ij}}\right)^2$




SUBROUTINE START
st:=1
δ := 2
go to FCN to evaluate $\varphi_i(y_0)$
$\varphi_i(y_0) \geq 0$?
i=1,...,m
NO
y_0 not feasible error return.
YES
go to STERTE with dec to evaluate X, YCT
return to main program


SUBROUTINE FCN
i:=1
sup:=10^{36}
$\varphi_i(x) = e_i - \sum\left(\frac{x_j - a_{ij}}{b_{ij}}\right)^2$
$\varphi_i(x) < 0$
YES
return with smallest value of $\varphi_i(x) = \varphi_{i_0}(x)$
NO
i:=i+1
φ_i < sup ?
YES
i_0:= i
sup := φ_i
NO
i ≤ m
YES
NO
return to calling program

Bibliography

1. Bellman, Richard: ed. Math. Optimization Techniques University of California Press, 1963.

2. Bledsoe, W. W.: The Use of Biological Concepts in the Analytical Study of Systems Panoramic Research, Incorporated, 1961.

3. Bledsoe, W. W.: Lethally Dependent Genes Using Instant Selection Panoramic Research, Incorporated, 1961.

4. Bremermann, H. J.: The Evolution of Intelligence ONR Technical Report No. 1, Contract Nonr 477(17), 1958.

5. Bremermann, H. J.: Limits of Genetic Control I.E.E.E. Transactions, Prof. Group. Mil. Electronics, April, 1963.

6. Bremermann, H. J.: Non Genetic Control in Biological Systems Technical Report, Contracts Nonr 222(85) and Nonr 3656(08), 1963.

7. Bremermann, H. J., and Salaff, S.: Experiments with Patterns of Evolution Technical Report, Contracts Nonr 222(85) and Nonr 3656(08), 1963.

8. Charnes, A., Cooper, W. W., and Henderson, A: An Introduction to Linear Programming Wiley, 1953.

9. Curtiss, J.: A Theoretical Comparison of the Efficiencies of Two Classical and a Monte Carlo Method for Computing One Component of the Solution of a Set of Linear Algebraic Equations in Symposium on Monte Carlo Methods, Wiley, 1954.

10. Dantzig, G. B.: Linear Programming and Extensions Princeton University Press, 1963.

11. Friedburg, R. M.: A Learning Machine IBM J. Res. and Dev., part I, vol. 2, pp. 2-13, January 1958, part II, vol. 5, pp. 183-191, July 1959.

12. Gale, D.: The Theory of Linear Economic Models
McGraw-Hill, 1962.

13. Gelfand, I. M., and Tsetlin, M. L.: Some Methods of Control for Complex Systems
Russian Math. Surveys, vol. 17, no. 1, 1961.

14. Hadley, G.: Linear Programming
Addison Wesley, 1962.

15. Hooke, R., and Jeeves, T.: "Direct Search" Solution of Numerical and Statistical Problems
Journal of the A. C. M., April 1961.

16. Minski, M.: Steps Toward Artificial Intelligence
Proc. I. R. E., vol. 49, pp. 3-30, January 1961.

17. Stern, C.: Principles of Human Genetics
2nd ed., Freeman, 1960.

18. Stone, W. S.: The Dominance of Natural Selection and the Reality of Superspecies (Species Groups) in the Evolution of Drosophila
Studies in Genetics, University of Texas, Publication No. 6215, March 1962.

19. Von Neumann, J., and Goldstine, H. H.: Numerical Inverting of Matrices of High Order
Bulletin of the A.M.S., November 1947.

20. Wilde, Optimum Seeking Methods
Prentice Hall, Englewood Cliffs, New Jersey, 1963.